Import needed modules

```
!pip install tensorflow==2.9.1
```

Show hidden output

```
# import system libs
import os
import time
import shutil
import pathlib
import itertools

# import data handling tools
import cv2
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

# import Deep learning Libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")

print ('modules loaded')
```

```
modules loaded
```

Data Preprocessing

Read data and store it in dataframe

```
# Generate data paths with labels
data_dir = 'chest-xray-pneumonia/chest_xray/train'
filepaths = []
labels = []

folds = os.listdir(data_dir)
for fold in folds:
    foldpath = os.path.join(data_dir, fold)
    filelist = os.listdir(foldpath)
    for file in filelist:
        fpath = os.path.join(foldpath, file)
        filepaths.append(fpath)
        labels.append(fold)

# Concatenate data paths with labels into one dataframe
Fseries = pd.Series(filepaths, name= 'filepaths')
Lseries = pd.Series(labels, name='labels')
df = pd.concat([Fseries, Lseries], axis= 1)


df
```

Split dataframe into train, valid, and test

```
# train dataframe
train_df, dummy_df = train_test_split(df,  train_size= 0.8, shuffle= True, random_state= 123)

# valid and test dataframe
valid_df, test_df = train_test_split(dummy_df,  train_size= 0.6, shuffle= True, random_state= 123)
```

Create image data generator

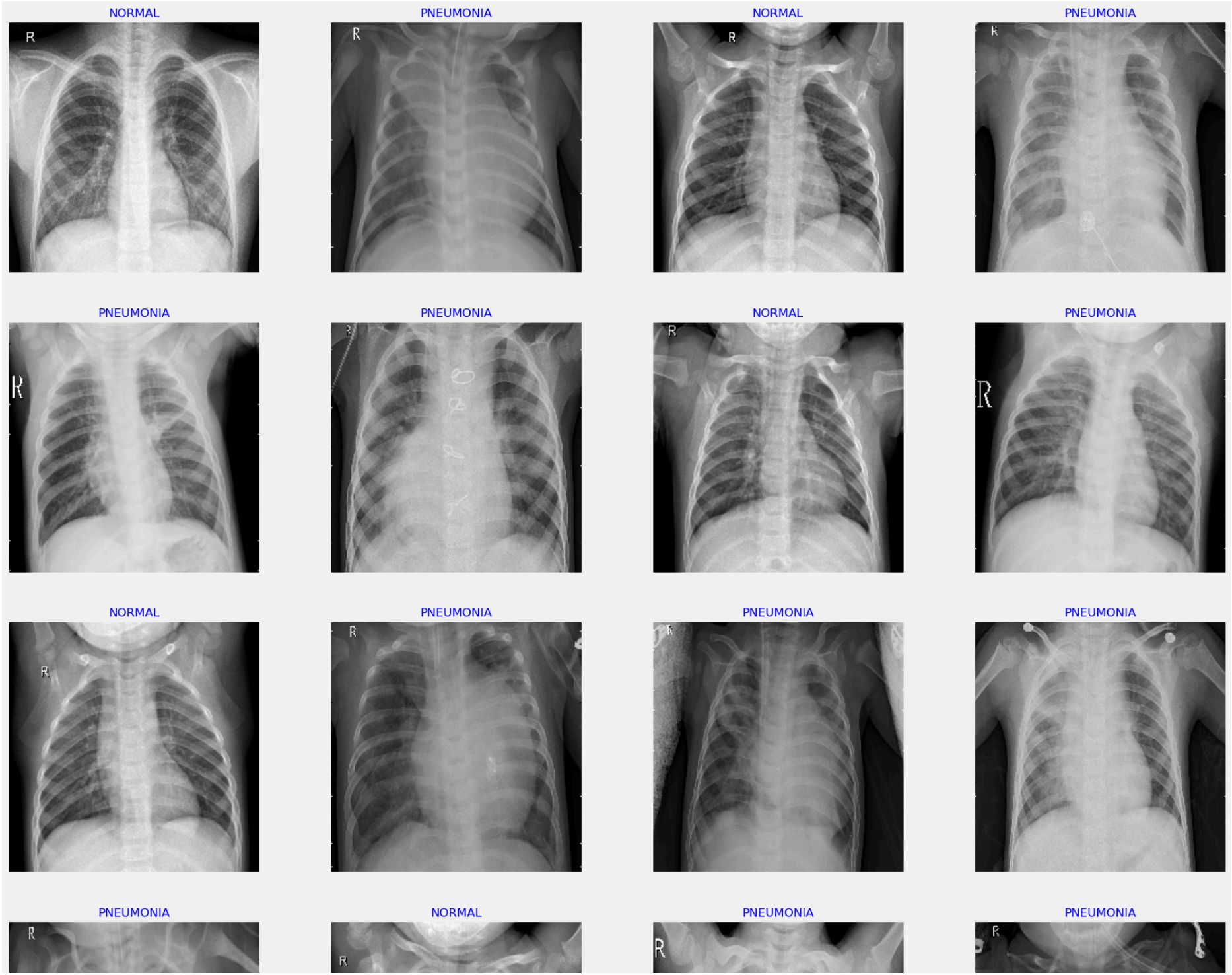
```
# crobed image size
batch_size = 16
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

# Recommended : use custom function for test data batch size, else we can use normal batch size.
ts_length = len(test_df)
test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size

# This function which will be used in image data generator for data augmentation, it just take the image and return it again.
def scalar(img):
    return img
```

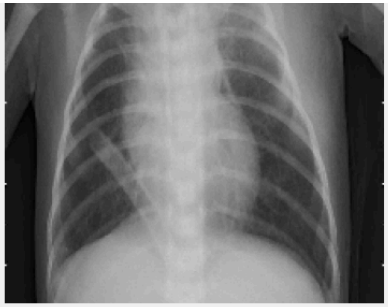
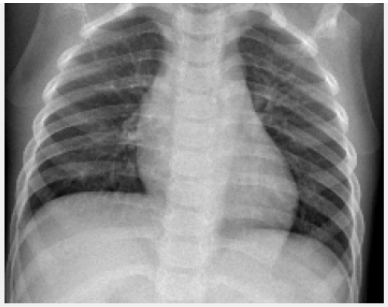
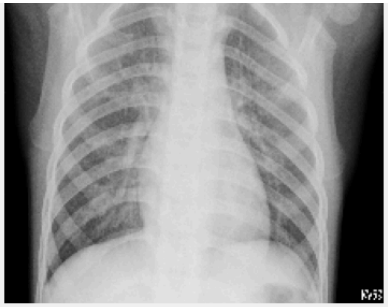
```
tr_gen = ImageDataGenerator(preprocessing_function= scalar)
ts_gen = ImageDataGenerator(preprocessing_function= scalar)

train_gen = tr_gen.flow_from_dataframe( train_df, x_col= 'filepaths', y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                    color_mode= 'rgb', shuffle= True, batch_size= batch_size)

valid_gen = ts_gen.flow_from_dataframe( valid_df, x_col= 'filepaths', y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                    color_mode= 'rgb', shuffle= True, batch_size= batch_size)

# Note: we will use custom test_batch_size, and make shuffle= false
test_gen = ts_gen.flow_from_dataframe( test_df, x_col= 'filepaths', y_col= 'labels', target_size= img_size, class_mode= 'categorical',
                                    color_mode= 'rgb', shuffle= False, batch_size= test_batch_size)
```

```
Found 4172 validated image filenames belonging to 2 classes.
Found 626 validated image filenames belonging to 2 classes.
Found 418 validated image filenames belonging to 2 classes.
```

Show sample from train data

```
g_dict = train_gen.class_indices      # defines dictionary {'class': index}
classes = list(g_dict.keys())       # defines list of dictionary's kays (classes), classes names : string
images, labels = next(train_gen)      # get a batch size samples from the generator

plt.figure(figsize= (20, 20))

for i in range(16):
    plt.subplot(4, 4, i + 1)
    image = images[i] / 255       # scales data to range (0 - 255)
    plt.imshow(image)
    index = np.argmax(labels[i])  # get image index
    class_name = classes[index]   # get class of image
    plt.title(class_name, color= 'blue', fontsize= 12)
    plt.axis('off')
plt.show()
```


NORMAL
PNEUMONIA
NORMAL
PNEUMONIA
PNEUMONIA
PNEUMONIA
NORMAL
PNEUMONIA
NORMAL
PNEUMONIA
NORMAL
PNEUMONIA
PNEUMONIA
PNEUMONIA
PNEUMONIA
NORMAL
PNEUMONIA
PNEUMONIA


Model Structure

Generic Model Creation

```
# Create Model Structure
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)
class_count = len(list(train_gen.class_indices.keys())) # to define number of classes in dense layer

# create pre-trained model (you can built on pretrained model such as :  efficientnet, VGG , Resnet )
# we will use efficientnetb3 from EfficientNet family.
base_model = tf.keras.applications.efficientnet.EfficientNetB0(include_top= False, weights= "imagenet", input_shape= img_shape, pooling= 'max')
# base_model.trainable = False

model = Sequential([
    base_model,
    BatchNormalization(axis= -1, momentum= 0.99, epsilon= 0.001),
    Dense(256, kernel_regularizer= regularizers.l2(l= 0.016), activity_regularizer= regularizers.l1(0.006),
                bias_regularizer= regularizers.l1(0.006), activation= 'relu'),
    Dropout(rate= 0.45, seed= 123),
    Dense(class_count, activation= 'softmax')
])

model.compile(Adamax(learning_rate= 0.001), loss= 'categorical_crossentropy', metrics= ['accuracy'])

model.summary()
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 efficientnetb0 (Functional)  (None, 1280)            4049571

 batch_normalization (BatchN  (None, 1280)            5120
 ormalization)

 dense (Dense)               (None, 256)             327936

 dropout (Dropout)           (None, 256)             0

 dense_1 (Dense)             (None, 2)               514

=================================================================
Total params: 4,383,141
Trainable params: 4,338,558
Non-trainable params: 44,583
_________________________________________________________________
```

Train model

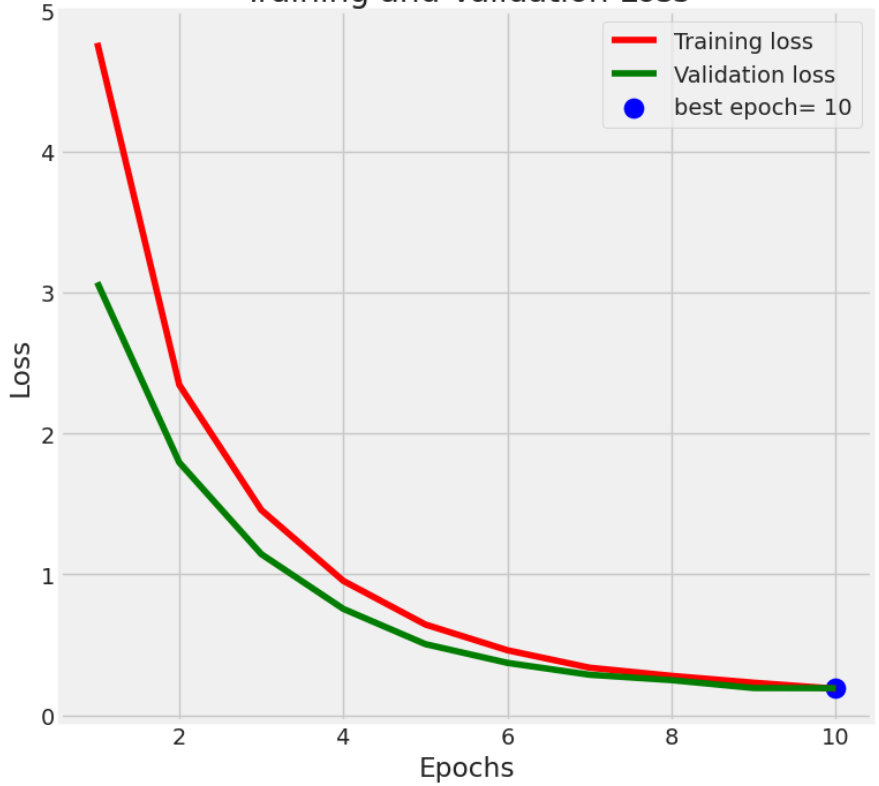
```
batch_size = 16   # set batch size for training
epochs = 10   # number of all epochs in training

history = model.fit(x= train_gen, epochs= epochs, verbose= 1, validation_data= valid_gen,
                    validation_steps= None, shuffle= False)
```

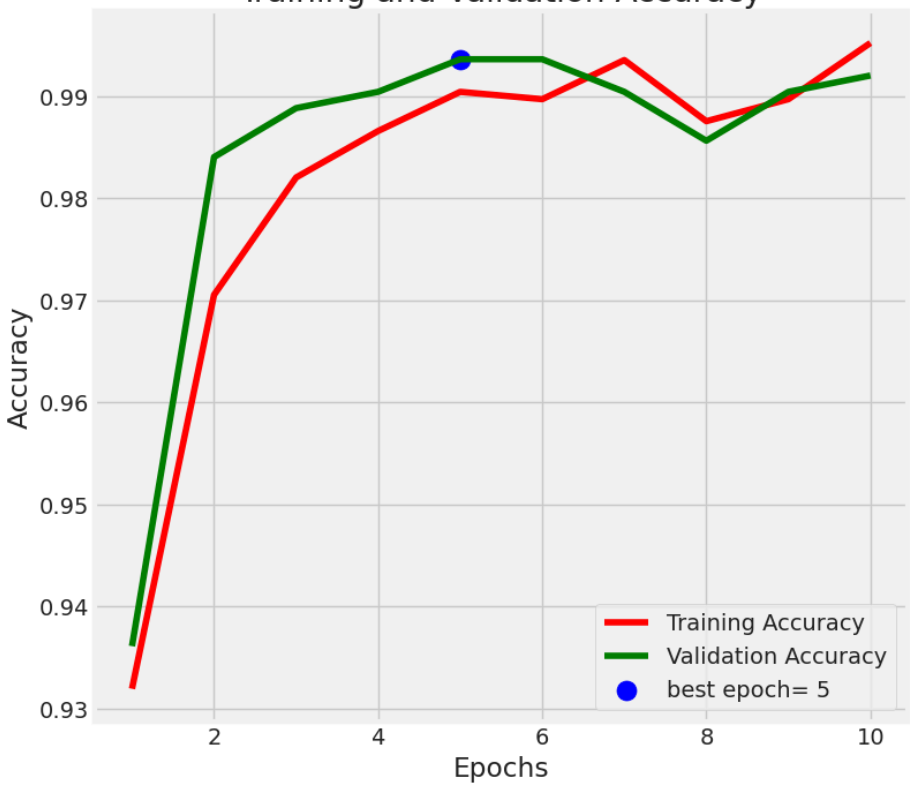
```
Epoch 1/10
261/261 [==============================] - 127s 434ms/step - loss: 4.7730 - accuracy: 0.9319 - val_loss: 3.0728 - val_accuracy: 0.9361
Epoch 2/10
261/261 [==============================] - 71s 271ms/step - loss: 2.3450 - accuracy: 0.9705 - val_loss: 1.7940 - val_accuracy: 0.9840
Epoch 3/10
261/261 [==============================] - 70s 267ms/step - loss: 1.4567 - accuracy: 0.9820 - val_loss: 1.1433 - val_accuracy: 0.9888
Epoch 4/10
261/261 [==============================] - 70s 269ms/step - loss: 0.9532 - accuracy: 0.9866 - val_loss: 0.7545 - val_accuracy: 0.9904
Epoch 5/10
261/261 [==============================] - 71s 271ms/step - loss: 0.6435 - accuracy: 0.9904 - val_loss: 0.5044 - val_accuracy: 0.9936
Epoch 6/10
261/261 [==============================] - 70s 268ms/step - loss: 0.4609 - accuracy: 0.9897 - val_loss: 0.3713 - val_accuracy: 0.9936
Epoch 7/10
261/261 [==============================] - 70s 269ms/step - loss: 0.3373 - accuracy: 0.9935 - val_loss: 0.2870 - val_accuracy: 0.9904
Epoch 8/10
261/261 [==============================] - 70s 268ms/step - loss: 0.2794 - accuracy: 0.9875 - val_loss: 0.2488 - val_accuracy: 0.9856
Epoch 9/10
261/261 [==============================] - 70s 266ms/step - loss: 0.2306 - accuracy: 0.9897 - val_loss: 0.1927 - val_accuracy: 0.9904
Epoch 10/10
261/261 [==============================] - 70s 269ms/step - loss: 0.1878 - accuracy: 0.9952 - val_loss: 0.1916 - val_accuracy: 0.9920
```

Display model performance

```
# Define needed variables
tr_acc = history.history['accuracy']
tr_loss = history.history['loss']
```

```
val_acc = history.history['val_accuracy']
val_loss = history.history['val_loss']
index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]
index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]
Epochs = [i+1 for i in range(len(tr_acc))]
loss_label = f'best epoch= {str(index_loss + 1)}'
acc_label = f'best epoch= {str(index_acc + 1)}'

# Plot training history
plt.figure(figsize= (20, 8))
plt.style.use('fivethirtyeight')

plt.subplot(1, 2, 1)
plt.plot(Epochs, tr_loss, 'r', label= 'Training loss')
plt.plot(Epochs, val_loss, 'g', label= 'Validation loss')
plt.scatter(index_loss + 1, val_lowest, s= 150, c= 'blue', label= loss_label)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(Epochs, tr_acc, 'r', label= 'Training Accuracy')
plt.plot(Epochs, val_acc, 'g', label= 'Validation Accuracy')
plt.scatter(index_acc + 1 , acc_highest, s= 150, c= 'blue', label= acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout
plt.show()
```





Evaluate model

```
ts_length = len(test_df)
test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n == 0 and ts_length/n <= 80]))
test_steps = ts_length // test_batch_size

train_score = model.evaluate(train_gen, steps= test_steps, verbose= 1)
valid_score = model.evaluate(valid_gen, steps= test_steps, verbose= 1)
test_score = model.evaluate(test_gen, steps= test_steps, verbose= 1)

print("Train Loss: ", train_score[0])
print("Train Accuracy: ", train_score[1])
print('-' * 20)
print("Validation Loss: ", valid_score[0])
print("Validation Accuracy: ", valid_score[1])
print('-' * 20)
```

```
print("Test Loss: ", test_score[0])
print("Test Accuracy: ", test_score[1])
```

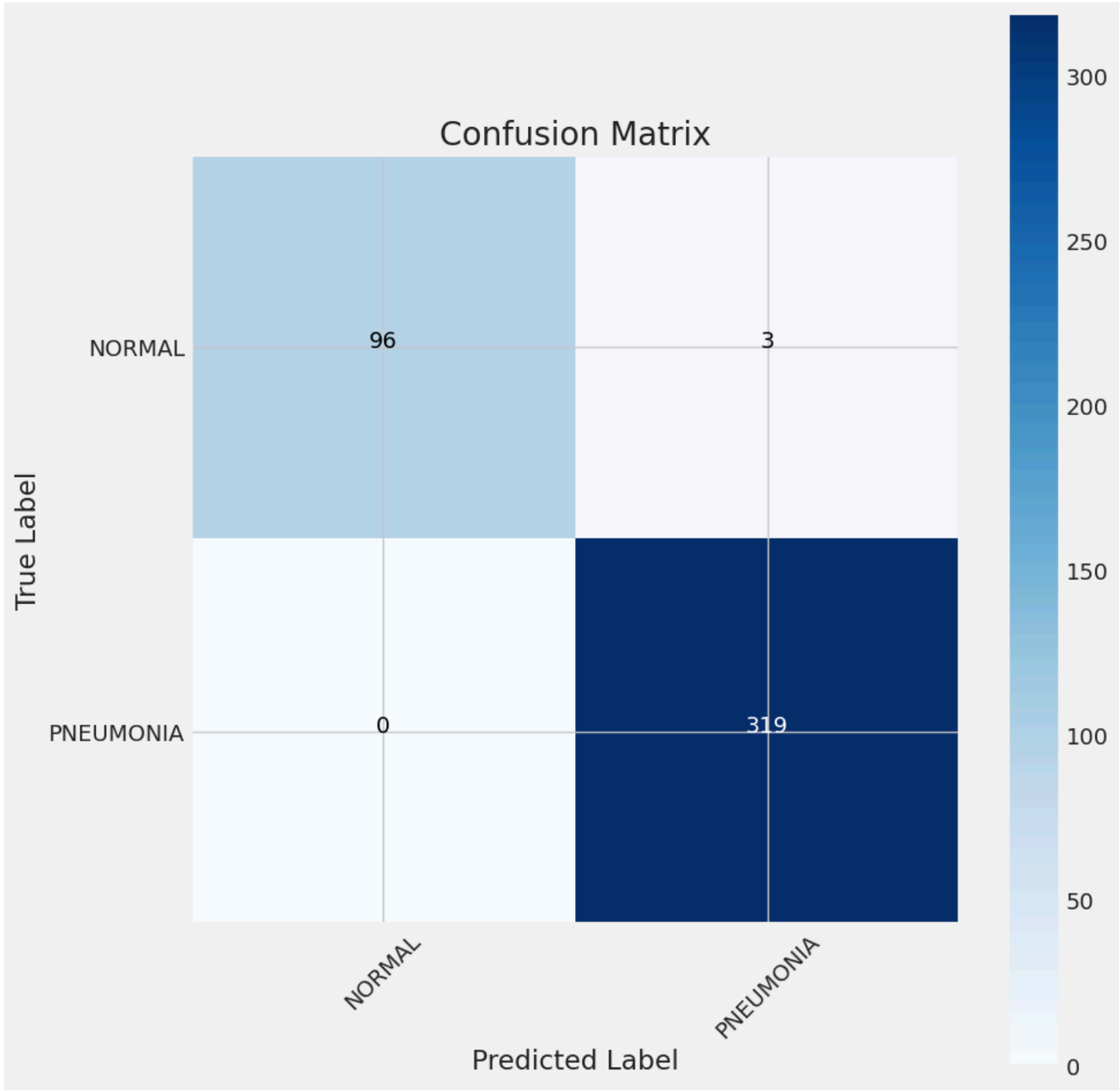
```
11/11 [==============================] - 2s 200ms/step - loss: 0.1767 - accuracy: 1.0000
11/11 [==============================] - 2s 201ms/step - loss: 0.1893 - accuracy: 0.9943
11/11 [==============================] - 9s 839ms/step - loss: 0.1862 - accuracy: 0.9928
Train Loss:  0.1766764372587204
Train Accuracy:  1.0
--------------------
Validation Loss:  0.18926642835140228
Validation Accuracy:  0.9943181872367859
--------------------
Test Loss:  0.18622736632823944
Test Accuracy:  0.9928229451179504
```

Get Predictions

```
preds = model.predict_generator(test_gen)
y_pred = np.argmax(preds, axis=1)
```

Confusion Matrics and Classification Report

```
g_dict = test_gen.class_indices
classes = list(g_dict.keys())

# Confusion matrix
cm = confusion_matrix(test_gen.classes, y_pred)

plt.figure(figsize= (10, 10))
plt.imshow(cm, interpolation= 'nearest', cmap= plt.cm.Blues)
plt.title('Confusion Matrix')
plt.colorbar()

tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation= 45)
plt.yticks(tick_marks, classes)


thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, cm[i, j], horizontalalignment= 'center', color= 'white' if cm[i, j] > thresh else 'black')

plt.tight_layout()
plt.ylabel('True Label')
plt.xlabel('Predicted Label')

plt.show()
```


Confusion Matrix
NORMAL
PNEUMONIA
True Label
96
3
0
319
NORMAL
PNEUMONIA
Predicted Label
300
250
200
150
100
50
0

```
# Classification report
print(classification_report(test_gen.classes, y_pred, target_names= classes))
```

```
              precision    recall  f1-score   support

      NORMAL       1.00      0.97      0.98        99
   PNEUMONIA       0.99      1.00      1.00       319

    accuracy                           0.99       418
   macro avg       1.00      0.98      0.99       418
weighted avg       0.99      0.99      0.99       418
```

Save model

```
model_name = model.input_names[0][:-6]
subject = 'Indian Rock Python'
acc = test_score[1] * 100
save_path = ''

# Save model
save_id = str(f'{model_name}-{subject}-{"%.2f" %round(acc, 2)}.h5')
model_save_loc = os.path.join(save_path, save_id)
model.save(model_save_loc)
print(f'model was saved as {model_save_loc}')

# Save weights
weight_save_id = str(f'{model_name}-{subject}-weights.h5')
weights_save_loc = os.path.join(save_path, weight_save_id)
model.save_weights(weights_save_loc)
print(f'weights were saved as {weights_save_loc}')
```

```
model was saved as efficientnetb0-Indian Rock Python-99.28.h5
weights were saved as efficientnetb0-Indian Rock Python-weights.h5
```

Generate CSV files containing classes indicies & image size

```
class_dict = train_gen.class_indices
```